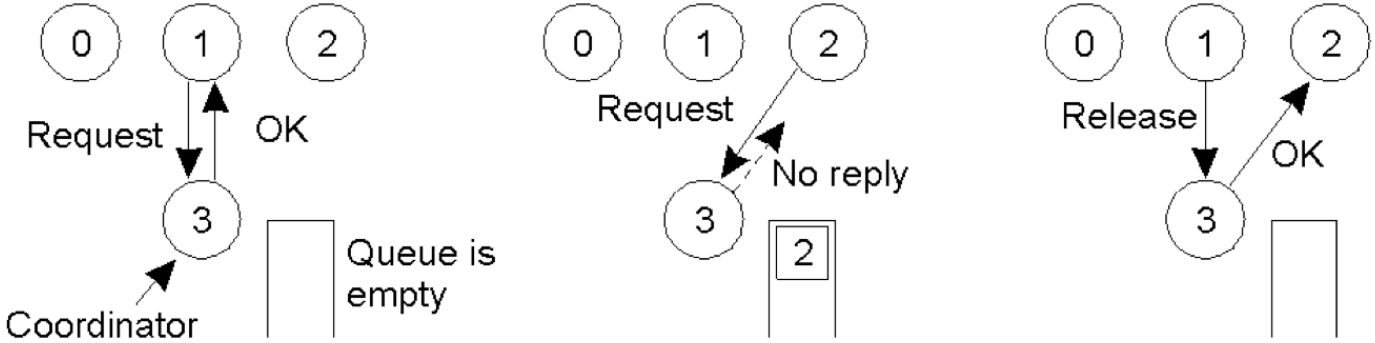
Discussion Section 4

Distributed Systems
Spring 2014

Problems Chapter 6

Centralized mutual exclusion

Q: In the centralized approach of mutual exclusion, upon receiving a message from a process releasing its exclusive access, the coordinator normally grants permission to the first process in the queue. Give another algorithm for the coordinator.



A: Requests could be implemented with priority levels. In such cases, the coordinator will grant access to the request with highest priority

Ricart and Agrawala's Algorithm

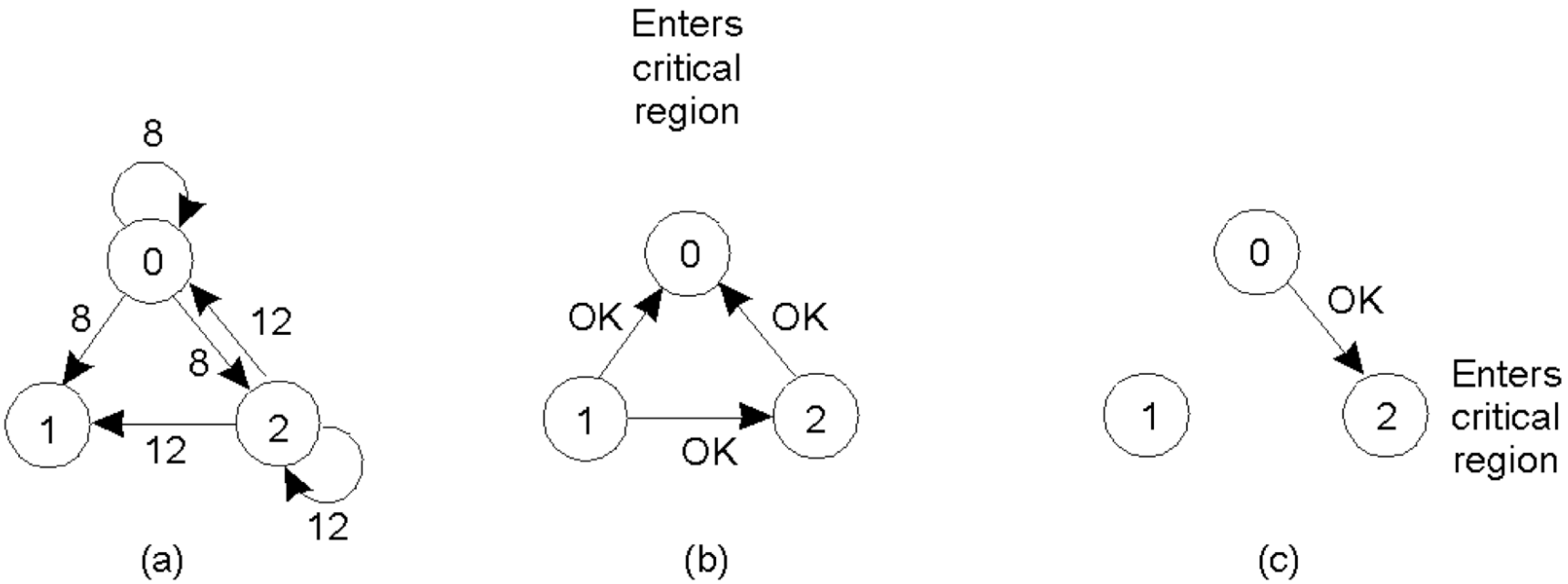
If a process want to access a shared resource, it build a message with the resource name, its process number and a (logical) timestamp. And sends it to all the processes including itself.

1. If the receiver is not accessing the resource and does not want to access it, it sends back an OK message to the sender
2. If the receiver already has access the resource, it simply does not reply. Instead, it queues request
3. If the receiver wants to access the resource but has not done so, it compares the timestamps and the lowest one wins. If the incoming message wins, the receiver sends back an OK message.

After sending a request, a process needs to wait for an OK from all the other processes to be able to access the shared resource.

Ricart and Agrawala's Algorithm

Example:



Problems Chapter 6

Ricart and Agrawala's Algorithm

Q: A distributed system may have multiple independent resources. Imagine that process 0 wants to access resource A and process 1 wants to access resource B. Can Ricart and Agrawala's algorithm lead to deadlocks?

A: Depends on whether or not resource accesses are strictly sequential. The procedure will be lock-free if the processes are allowed to use a single resource at a time.

Potential deadlock:

P1 access A and (without releasing A) tries to access B

P2 access B and (without releasing B) tries to access A

Problems Chapter 7

Weak consistency models

Q: Explain in your own words what the main reason is for actually considering weak consistency models.

A: For performance and scalability reasons

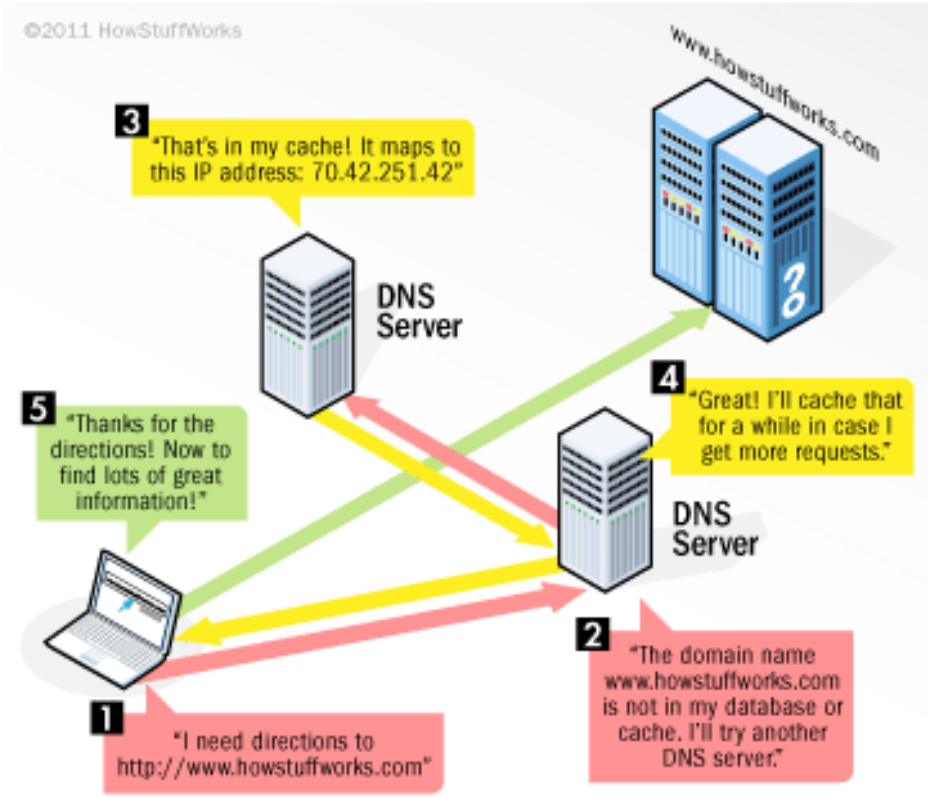
Weak consistency models come from the need to replicate for performance. However, efficient replication can be done only if we can ***avoid global synchronizations****, which, in turn, can be achieved by* ***loosening consistency constraints***

Problems Chapter 7

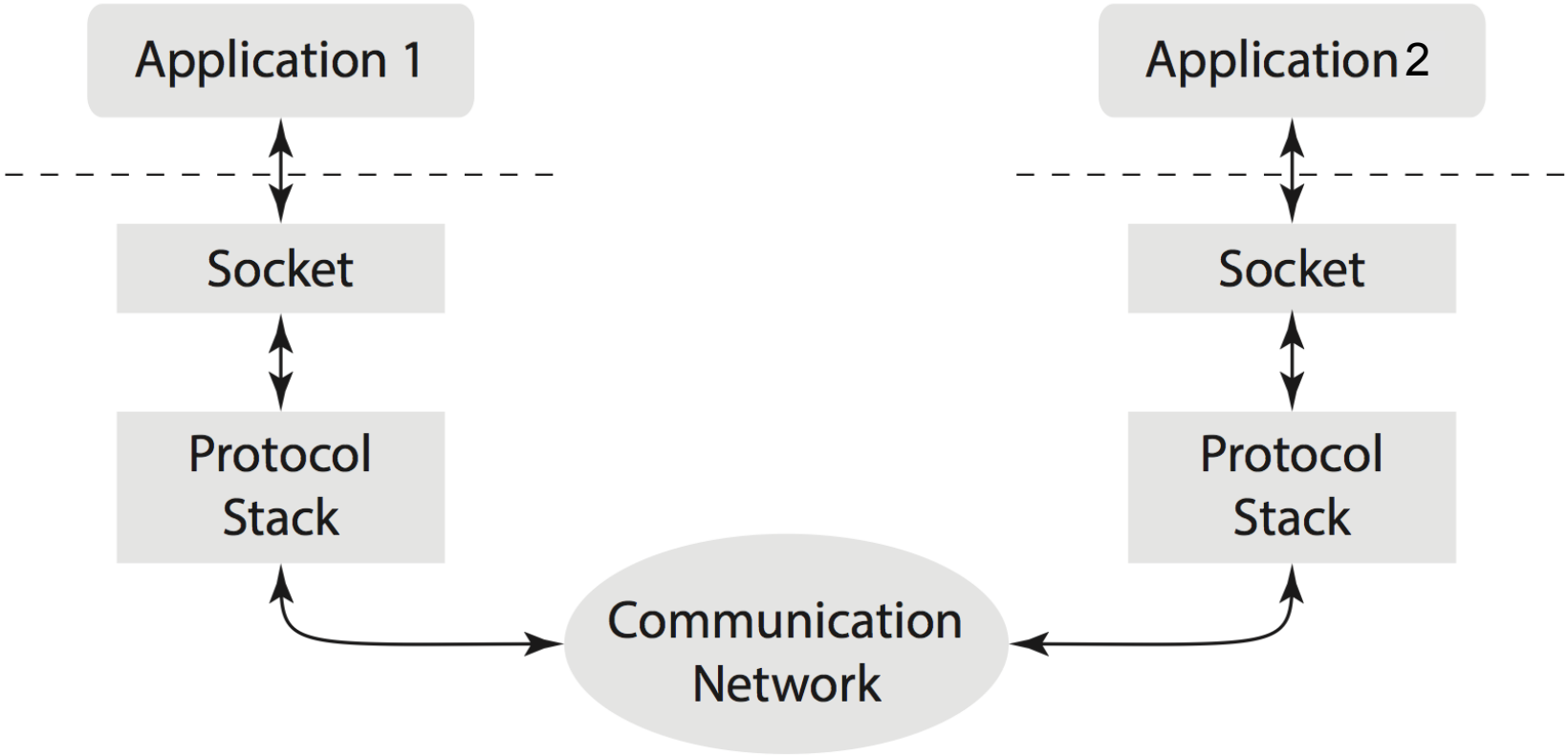
Weak consistency models

Q: Explain how replication in DNS takes place, and why it actually works so well

A: The basic idea is that name servers ***cache previously looked*** *up results. These results can be kept in a cache for a long time, because DNS makes the* ***assumption*** *that* ***name-to-address mappings do not change often****.*



Sockets


Application 1
Application 2
Socket
Socket
Protocol Stack
Protocol Stack
Communication Network

Sockets

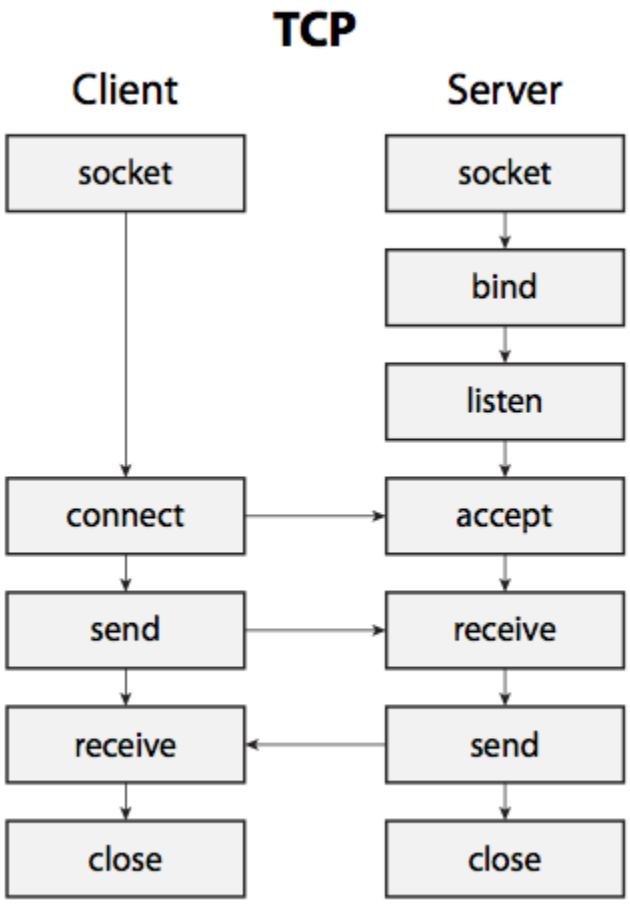
- Interface between application layer and the transport layer

- Two types :
  - TCP : Connection-oriented
  - UDP : Connectionless

Important System Calls

- socket()
- bind()
- listen()
- accept()
- connect()
- send() and recv() (or read() and write())
- sendto() and recvfrom()
- close() and shutdown()

TCP Client/Server



socket()

- Called by both client and server
- When you make a call to socket(), it returns a socket descriptor
- socket (family, type, protocol)
  - **family** - IPv4 (AF_INET) or IPv6 (AF_INET6)
  - **type** - SOCK_DGRAM (datagrams) or SOCK_STREAM(stream)
  - **protocol** - set it to 0 to let the socket choose the correct protocol based on type

bind()

- Used to associate a socket with a port on the local machine
- Port number is used by the kernel to match the incoming packet to a process (e.g. HTTP - 80)
- Not all ports can be used :
  - 0-1024 : Well-known ports
  - 1025-65535 : can be used

listen()

- Called by servers
- Allows server to prepare the socket for incoming connections
- Puts the server in a passive mode ready to accept connections
- listen() is non-blocking, returns immediately
- We need to call bind() before calling listen(), i.e., a port needs to be specified

accept()

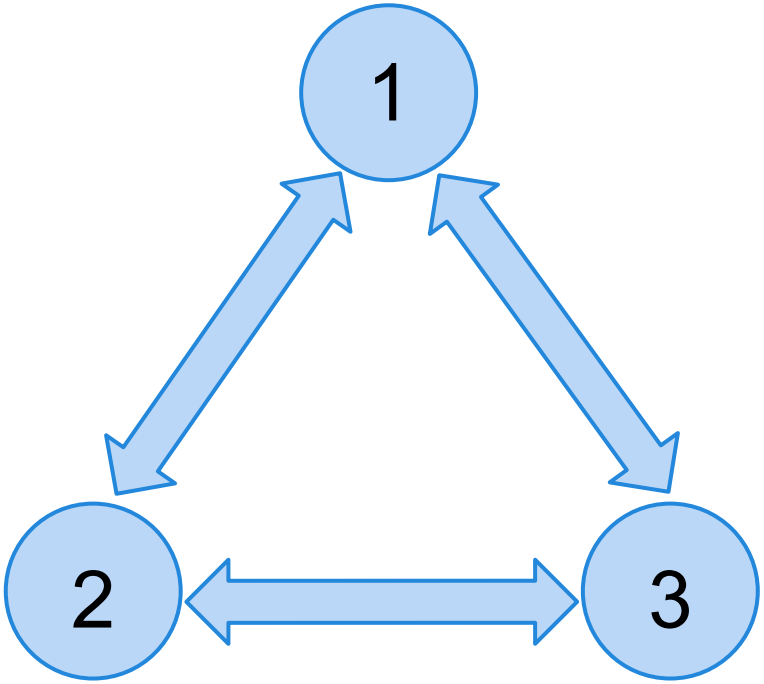
- Called by servers
- Server waits for a connection
- On receiving a connection, system creates a new socket and returns the descriptor
- accept() is blocking, waits for connection before returning

connect()

- Called by clients
- Binds a destination to the socket
- Changes the socket state from unconnected to connected

Assignment

- None of them is just a server or a client, they need to be both
- You can use two sockets : one for listening and the other for connecting



Code

Server

Client